Vol.2019-IFAT-134 No.4
Vol.2019-DC-112 No.4
2019/3/22

ドキュメントビルダ Sphinx による運用手順書構造化への取り組み

波田野 裕一[†1]

概要：IT システム運用現場において，そのサービスおよび運用の品質を担保するために数多くの運用手順書が作成され，利用されている．これらの運用手順書は，各運用現場で定められた独自フォーマットに従って各担当者が自己の経験に基づいた独自の手法で作成され，かつ非構造的な記述により作業者にとって扱いづらいものとなっていることが少なくない． 本発表では，オープンソースのドキュメントビルダ Sphinx の特性を紹介し，Sphinx を活用した運用手順の構造化手法およびテキストファイルベースの運用手順書管理手法について述べる．

キーワード：システム運用，運用手順書，運用ドキュメント，テクニカルライタ

1. はじめに

社会的重要性を増す IT システム運用において，サービス品質や運用品質を担保するために多様な運用ドキュメントが作成され，その維持に相当の工数を要している． その中でも，運用作業のさまざまな手順を記述した運用手順書は，運用品質の適正化および要員のスキル平準化を目的に，多大な工数を掛けて整備され，利用されている． しかし，多くの運用手順書は，各運用現場で定められた独自フォーマットに従って，各担当者が自己の経験に基づいた独自の手法で作成しており，非構造的な記述によって作業者には扱いづらいものとなっていることが少なくない．

本論文では，オープンソースのドキュメントビルダ Sphinx の特性を紹介し，Sphinx を活用した運用手順の構造化手法およびテキストファイルベースの運用手順書管理手法について述べる．

2. ドキュメントビルダ Sphinx

Sphinx[1]は，「知的で美しいドキュメントを簡単に作れるようにする」ことを目的に開発されたツールで，プログラミング言語 Python で実装されている．当初はプログラミング言語 Python の公式ドキュメントを構築するためのツールとして開発が進められたが，現在は Web サイトや論文，一般的なドキュメントなど幅広いドキュメントの作成に利用されている．

Sphinx には以下の特徴がある．

ワンソース・マルチアウトプット：1 つのドキュメントソースを基に，以下の形式の媒体に出力することができる．

- HTML
- LaTeX (PDF)
- ePub
- Texinfo
- man
- プレーンテキスト (markdown に近似)

多彩なインクルード：他のドキュメントやソースコード，CSV ファイルなどをドキュメント内にインクルードすることができる．ソースコードについては自動的にハイライトし，CSV ファイルについてはテーブル表示することができる．

多彩な自動リンク・インデックス作成機能：簡単にドキュメントツリーを定義し，隣接階層のドキュメントに対して自動的にリンクを作成することができる． また，意味的なマークアップや関係のある情報(引用，用語，関数，クラスなど)に対して自動的にリンクを作成することができ，更に，全体インデックスや，プログラミング言語特有のモジュールインデックスなども自動的に作成することができる．

3. マークアップ言語 reStructuredText

Sphinx は，マークアップ言語として reStructuredText[2]を使用し，そのパーサとしては Docutils[3]を使用する．

reStructuredText には以下の特徴がある．

プレーンテキストである：多くのマークアップ言語に共通の特徴であるが，更新差分を明示しやすいため，バージョン管理システムと非常に相性が良いと言える．

言語仕様が統一的である：reStructuredText は，Docutils プロジェクトによって言語仕様の定義とパーサの開発が行われているため言語仕様の把握が容易である．マークアップ言語として人気のある markdown には，類似実装が乱立していることとは対照的である．

非エンジニアにとって違和感が少なく可読性が高い：reStructuredText は，最も基本的な文法だけを使用して記述した場合，非エンジニアにとって違和感が少なく，そのままメールや議事録に貼付することができる形式となっている．(図 1)

†1 北陸先端科学技術大学院大学

Vol.2019-IFAT-134 No.4
Vol.2019-DC-112 No.4
2019/3/22

```
==========
見出し1
==========

:日時: 2019-03-22 13:00 - 13:30
:場所: 白山オフィス
:参加者: 小川，戸隠，湯本，清水川，波田野

sphinx
  Pythonで実装されたドキュメントビルダです．

----------
見出し2
----------

いくつかSphinxのサンプルを記述していきます．
まずは基本的なマークアップから．

- リストA
- リストB

  - リストB.a

2. 2番目から初めてもよい．
3. 3番目です．

見出し3
++++++++++++++++++++++++++++++

コマンド::

  ls -a ~

結果(例)::

  .exrc
```

図 1. reStructuredText の例

図 2. HTML 出力例

このことは，ドキュメント関係者の範囲が広範に渡る場合にも可読性が高いままに共有できることとなり，ドキュメント記述の再利用性が高くなることにもつながる．

reStructuredText では，基本文法で表現できない記述については，以下の機能で表現することになっている．

ディレクティブ： 主にブロックレベルでの特殊な表現を実現するための機能である．[4]

```
.. csv-table:: Frozen Delights!
   :file: ./frozen-delights.csv
```

図 3. ディレクティブの例 (CSV によるテーブル)

ロール： 主にインラインレベルでの特殊な表現を実現するための機能である．[5]

```
The area of a circle is :math:`A_\text{c} = (\pi/4) d^2`.
```

図 4. ロールの記述例 (TeX の埋め込み)

The area of a circle is $A_c = (\pi/4)d^2$.

図 5. HTML 出力例

また，特殊なディレクティブとして，replace ディレクティブがある． これは，インライン上に被置換文字を記述しておき，別途 replace ディレクティブで置換内容を宣言することで，パース時に置換を実施してくれるものである．

```
作業前に |server| にSSHログインします．
```

図 6. replace の例 (インライン部)

```
.. |server| replace:: login01.example.jp
```

図 7. replace の例(宣言部)

作業前に login01.example.jp にSSHログインします．

図 8. HTML 出力例

テンプレートドキュメントのインライン部分に被置換文字を記述しておき，出力するドキュメントの用途毎に置換内容を変更することで，使用状況に合わせた最適なドキュメントを生成することができる．

4. Sphinx 独自の拡張

Sphinx は，Docutils のパース結果を基にドキュメントをビルドするが，reStructuredText 内に Sphinx に対する指示が記述されている場合は，その指示に応じてドキュメントの加工を行う．

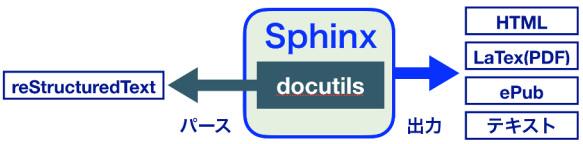

図 9. Sphinx と Docutils

reStructuredText 自体は，単一ページのドキュメントを記述するものであるため，Sphinx 独自の拡張としては以下のものがある．

- 複数ページに対応するための機能
- ページ間のリンクに関する機能
- Sphinx を独自拡張するための機能

ここでは，Sphinx の拡張のうち，複数の reStructuredText

Vol.2019-IFAT-134 No.4
Vol.2019-DC-112 No.4
2019/3/22

ファイルを 1 つのドキュメントとして扱う場合に重要となる機能について紹介する．

4.1 複数ページに対応するための機能

ドキュメント全体のツリー構造を構築するために，toctree ディレクティブが Sphinx による拡張機能として提供されている．

toctree ディレクティブ：複数の reStructuredText ドキュメントファイルのツリー関係を記述するために使用する．対象となる reStructuredText ドキュメントファイルをワイルドカード指定した場合，Sphinx のパーサはファイルの移動を自動的に検知してドキュメントに反映することができる．

```
.. toctree::
   :maxdepth: 2

   ./*
```

図 10. toctree ディレクティブの例

toctree ディレクティブは，Sphinx がビルドするドキュメントの論理構造を決定するため，ドキュメントの論理構造設計に従って適切に記述する必要がある．

4.2 ページ間のリンクに関する機能

ドキュメント内部のリンク機能として，ref ロールが Sphinx による拡張機能として提供されている．

ref ロール：見出しに付与されたラベルに対して，ドキュメント内部からインラインでリンクするために使用する．インライン上に，リンク先の見出しがそのまま挿入されるため，リンクの可読性が向上する．リンク先の見出しが移動したり，見出し名が修正されたりした場合，Sphinx のパーサはこれらの変更を自動的に検知してドキュメントに反映することができる．

```
.. _heading2:

------------------------------------------------------------
見出し2
------------------------------------------------------------
```

図 11. 見出しラベルの例

```
詳細については， :ref:`heading2` を参照してください．
```

図 12. ref ロールの例 (インライン部)

詳細については， 見出し2 を参照してください．

図 13. HTML 出力例

ref ロールは, Sphinx がビルドするドキュメント内における論理的なネットワーク構造を表現するために利用することができる．

5. Sphinx を利用した運用手順書の作成

運用手順書の作成において，Sphinx の特徴を活かすことで，以下を実現することが可能となる．

5.1 実行可能なコードを手順に組み込むことができる

Sphinx は多彩なインクルード機能を提供しており，その一つである literalinclude ディレクティブを使用することで実行可能なコードを手順書に組み込むことができる．

以下の例では，シェルスクリプト化したコマンドを手順に組み込んでいる．

```
ls |grep ${FILE_TARGET}
```

図 14. check-file.sh(ソースコード)

```
コマンド：

  .. literalinclude:: ./check-file.sh
     :language: sh
```

図 15. literal-include ディレクティブの例

コマンド:

```
ls -a ~
```

図 16. HTML 出力例

手順書の全てのコマンド入力を実行可能なコードにより記述することによって，以下のようなメリットがある．

実行内容の曖昧さ排除：実行可能なコードでコマンド入力を記述するということは，コマンド入力における曖昧な指示や利用者による読み替えを前提とすることができないため, 実行内容について曖昧さを排除することにつながる．

実行結果の正確さ確保：実行可能なコードでコマンド入力を記述するということは，その実行結果も容易に取得できる可能性が高いため，正確な実行結果を手順書に反映できる可能性も高くなる． これにより，手順書起因の確認ミスを抑制できることが期待できる．

適切な自動化の準備：実行可能なコードでコマンド入力を記述するということは，手順書そのものが疑似的な自動実行スクリプトとなることを意味し，手順書内のコードを実際に実行・検証しながら作業全体の自動化を進めることが可能となる．

5.2 個別作業に特化した手順書を作成することができる

Sphinx は多彩なインクルード機能と強力な置換機能を提供しており，これらを使用することで，個別の手順書に最適化された手順書を作成することができる．

以下の例では，手順テンプレートファイルでシェル変数の宣言と実行結果を記述し，置換定義ファイルで置換内容を記述し，両者をドキュメントから参照している．

```
変数の宣言:

  .. parsed-literal::

    FILE_TARGET="\ |FILE_TARGET|\ "

コマンド:

  .. literalinclude:: ./check-file.sh
    :language: sh

結果(例):

  .. parsed-literal::

    |FILE_TARGET|
```

図 17. 手順テンプレートファイル(template.txt)

```
.. |FILE_TARGET| replace:: target2019.txt
```

図 18. 置換定義ファイル(_defines.txt)

```
.. include:: ./_defines.txt

.. include:: ./template.txt
```

図 19. 両者をドキュメント本体から参照

変数の宣言:

```
FILE_TARGET="target2019.txt"
```

コマンド:

```
ls |grep ${FILE_TARGET}
```

結果(例):

```
target2019.txt
```

図 20. HTML 出力例

例えば，年に 1 回ある作業の手順の場合，_defines.txt 内の文字列"target2019.txt"を修正することで, 新しい年に特化した手順書を作成することができ，作業者は年に相当する部分の読み替えという余計な作業をする必要がなくなる．

読み替え不要な手順書を作業者が利用することによって，以下のようなメリットがある．

実行結果の曖昧さ排除：読み替え不要な手順書を作業者が利用するということは，正確な実行結果を手順書に反映できるため，結果確認について曖昧さを排除することにつながる． 前項でも述べたが，これにより手順書起因の確認ミスを抑制できることが期待できる．

作業者の負担軽減：読み替え不要な手順書を作業者が利用するということは，作業上不要なノイズを除去することになるため，作業者の確認負担を軽減して作業に集中することを可能にする．これにより手順書起因の確認ミスをより抑制できることが期待できる．

適切な自動化の準備：読み替え不要な手順書を作業者が利用するということは，置換対象部分を極限まで限定することでその作業における重要なチェックポイントを特定できることを意味する． つまり，手順上でチェックポイントの特定および判定するロジックを適切に構築することができれば，自動化時の判定ロジックのプロトタイプが完成することになる．

5.3 作成した手順書パーツを自然な形で再利用できる

Sphinx は多彩なインクルード機能を提供しており，その一つである include ディレクティブを使用することで，reStructuredText で記述されたドキュメント資産を多様な形で再利用することができる．

前項では，"両者をドキュメント本体から参照"するファイルから，スクリプトや手順テンプレートファイル，置換定義ファイルをインクルードしたが, この"両者をドキュメント本体から参照"するファイル自体も他の reStructuredText から include ディレクティブでインクルードすることができる． Sphinx において多彩なインクルード機能を多段に活用することで, 再利用性を維持しつつ, 個別事情に最適化した手順書を作成することが可能となる．

作成した手順書パーツを自然な形で再利用できることにより，以下のようなメリットがある．

作業定型化の誘因増大：テンプレートとなる手順書の作成による曖昧さの排除，作業者の負担軽減，適切な自動化の準備などのメリットが明確になるため，作業手順の定型化の大きな誘因になる． これにより，作業手順書作成に必要な工数を確保する上でマネジメント層の了解を得やすくなる．

作業定型化の精度向上：テンプレートとなる手順書の作成を継続することにより，曖昧さの排除，適切な自動化の準備の 2 点において学習効果が働くため，手順および定型化作業の精度が向上する．手順の精度向上は運用作業品質の向上に直結し，定型化作業の精度向上は業務引き継ぎや作業自動化の容易さに直結する．

ドキュメント再利用の再帰的な拡大：インクルード対象となるドキュメントの増大は，ドキュメント作成の容易さや工数軽減を促進し，インクルードを行うドキュメントの増大は，類似ドキュメント作成時の参照を容易にし，新たなドキュメント価値を生むことに繋がる． これらは，相互に影響しあいながら，再帰的にドキュメント自体の価値と再利用性を拡大していくことに繋がる．

6. テキストファイルベースの運用手順書管理手法

Sphinx が提供する多様な機能を活用することで，手順書の作成負担を大きく軽減することができるが，一方で複雑な相互参照は保守工数の増大を招く可能性がある．特にインクルード機能は強力であるがために多用されることになるが，不規則なインクルードの使用はファイル間の関係を複雑にし，保守を困難にしがちである．

これを回避するためには，以下のような管理手法が必要となる．

インクルードする側とされる側を混在させない:
一般的にドキュメントの参照は，一方からの参照は保守工数が低く，双方からの相互参照は保守工数が高い．これは，保守すべき箇所の数だけに留まらず，相互参照では双方の修正が完了するまではドキュメントが不完全な状態になるため，修正作業が不可分になることによる負担も含まれる．このことからも,インクルード機能を利用する際は，インクルードする側(参照側)のドキュメントとインクルードされる側(被参照側)のドキュメントを峻別する必要がある． 実際には，この 2 つに単純に分類することは難しいため，以下の 3 つのグループに分類することになる．

- 参照ドキュメント
- 参照被参照両用ドキュメント
- 被参照ドキュメント

図 21. インクルードの参照関係

参照・被参照関係をライフサイクルによって階層化する:
一般的に，抽象度が高く，汎用性のある知識により書かれ，表現形式や記述内容に一定の法則(ルール)や体系を持っているドキュメントは構造化しやすく，被参照ドキュメントに向いていることが多い． 一方で，具体度が高く，特定の事情や文脈に特化して書かれ，表現形式や記述内容もその事情や文脈に合わせて書かれるドキュメントは構造化しにくく,結果として参照ドキュメントとなることが多い．

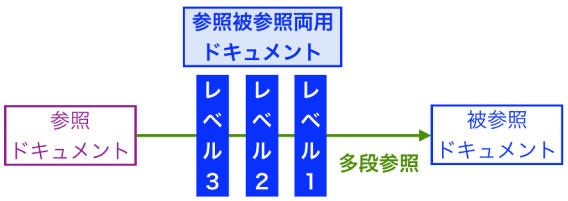

図 22. インクルード関係の階層化

被参照ドキュメントは寿命が長く，参照ドキュメントは寿命が短いため，そのライフサイクルに応じた保守工数を掛けることが合理的であると言える．限られた工数を適切に配分するためには，参照ドキュメントを使い捨て感覚で作成することが必要になる場合もある．

ドキュメントリポジトリを階層別，用途別に分ける: バージョン管理システム上のドキュメントリポジトリは，一般的には組織単位もしくは業務単位で作成することが多いが，特に被参照ドキュメントについては疎結合・分散を意識する必要がある． 大きなドキュメントリポジトリは，ライフサイクルの粒度を揃えることが難しく，更新履歴などが複雑で，管理工数が大きくなりがちである． ライフサイクルや目的にあわせて，可能な限り小規模なリポジトリを作成・保守することが望ましい．

7. おわりに

本論では，ドキュメントビルダ Sphinx の特性を中心に，運用手順書の作成およびテキストファイルベースの運用手順書管理手法の概要について紹介した．

本取り組みは，Amazon Web Services のユーザコミュニティの一つである JAWS-UG CLI 専門支部において実施しているものである． 同コミュニティは，4 年半の活動期間に 120 回以上のハンズオンを開催し，その手順書総数は 1000 を超える． このような短期間に，初心者でも迷わない手順書を大量に作成する必要があったことから，本論のような手法が誕生した．

インクルードによるソースコードベースの入力および置換による入出力情報の実際化は，運用手順書の利用者である作業者にとってわかりやすく，その作成者にとってもドキュメントの構造化を意識しやすくなるため，手順書の精度向上および運用作業品質の向上に極めて有効である．本論がその一助になれば幸いである．

参考文献

[1] "Sphinx Python Documentation Generator", http://www.sphinx-doc.org/, （参照 2019-02-27）.
[2] "reStructuredText Markup Syntax and Parser Component of Docutils", http://docutils.sourceforge.net/rst.html, （参照 2019-02-27）.
[3] "Docutils: Documentation Utilities Written in Python, for General- and Special-Purpose Use", http://docutils.sourceforge.net/index.html, （参照 2019-02-27）.
[4] "reStructuredText Directives", http://docutils.sourceforge.net/docs/ref/rst/directives.html, （参照 2019-02-27）.
[5] "reStructuredText Interpreted Text Roles", http://docutils.sourceforge.net/docs/ref/rst/roles.html, （参照 2019-02-27）.